

Wie können Softwaremetriken tatsächlich das Entwicklerleben verbessern?

Dr. Andreas Arnold (andreas.arnold@andrena.de)

Dr. Joachim Melcher (joachim.melcher@andrena.de)

Entwicklertag Karlsruhe, 9. Mai 2012

Metriken - Der erste Ansatz

- ▶ Idee: Software-Metriken könnten für Projekt nützlich sein
- ▶ Umsetzung:
 - ▶ viele Metriken werden automatisiert berechnet
 - ▶ Unsicherheit, was damit nun anzufangen ist
 - ▶ keine abgeleiteten Handlungen
- ▶ Konsequenz: Metriken werden nach kurzer Zeit ignoriert

Was sind Software-Metriken?

- ▶ statische Code-Analyse
- ▶ Grundlage Quelltext (bzw. abgeleiteter Kontroll-/Datenflussgraph)
- ▶ Quantifizierung einer **internen** Eigenschaft anhand konkreter Regeln

- ▶ eigentlich Interesse an **externen** Eigenschaften
- ▶ Software-Metrik als **Indikator** für externe Eigenschaft
- ▶ Abhängigkeit muss durch Experiment gezeigt werden (in der Praxis oft vernachlässigt!)

Was können Metriken?

- ▶ Vergleich über die Zeit oder mit anderen Projekten
- ▶ Indikator für Entwicklungstendenz bzw. aktuellen Zustand
- ▶ Hinweis auf mögliche Problemstellen
- ▶ Feststellen von Handlungsbedarf
- ▶ aber auch: Argumentationsgrundlage für Entwickler gegenüber Management

Es gibt viele Metriken

Auswahl einiger Metriken

- ▶ Lines of Code (LOC)
- ▶ zyklomatische Komplexität (McCabe)
- ▶ Depth of Inheritance Tree (DIT)
- ▶ Number of Children (NOC)
- ▶ Response For a Class (RFC)
- ▶ Weighted Methods per Class (WMC)
- ▶ Coupling Between Objects (CBO)
- ▶ Lack of Cohesion in Methods (LCOM)

Es gibt viele Metriken

Mögliche Klassifikation

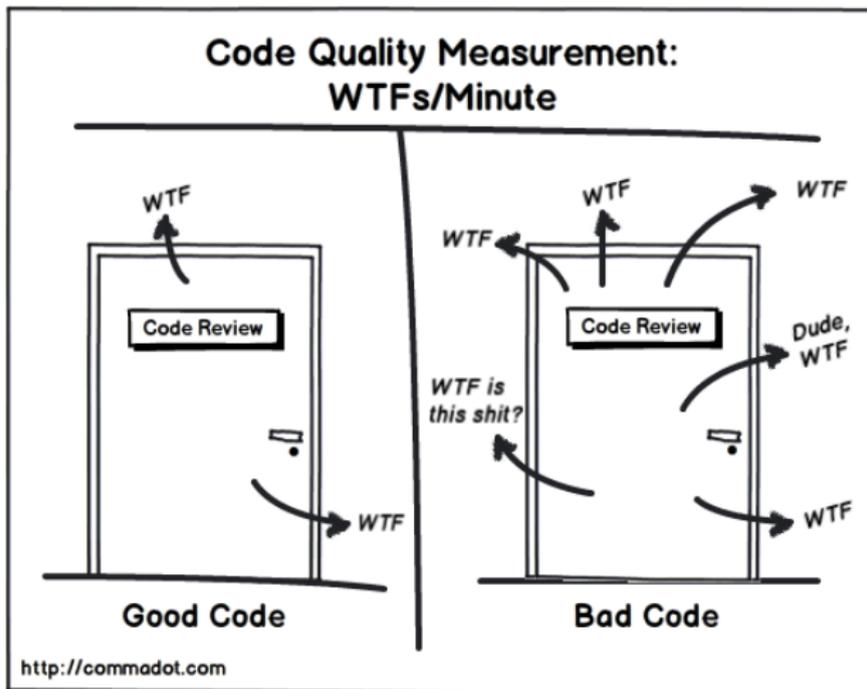
- ▶ traditionell
 - ▶ Größenmetriken
 - ▶ Komplexitätsmetriken
- ▶ objekt-orientiert
 - ▶ Methodenebene
 - ▶ Klassenebene
 - ▶ Vererbungshierarchie

Entscheidungskriterien

- ▶ Einfachheit (einfach zu messen/verstehen)
- ▶ Handlung ableitbar (einfach zu reparieren)
- ▶ Robustheit (Zusammenhang zwischen Metriken)

Sinnvolle Metriken

WTFs/min



Sinnvolle Metriken

ISIS/USUS-Metriken

- ▶ Testabdeckung
- ▶ Anzahl Pakete in Zyklen
- ▶ Average Component Dependency
- ▶ Anzahl Methoden mit zyklomatische Zahl größer 5
- ▶ Anzahl Klassen mit mehr als 20 Methoden
- ▶ Anzahl Methoden mit mehr als 15 LOC
- ▶ Anzahl Compiler-Warnungen

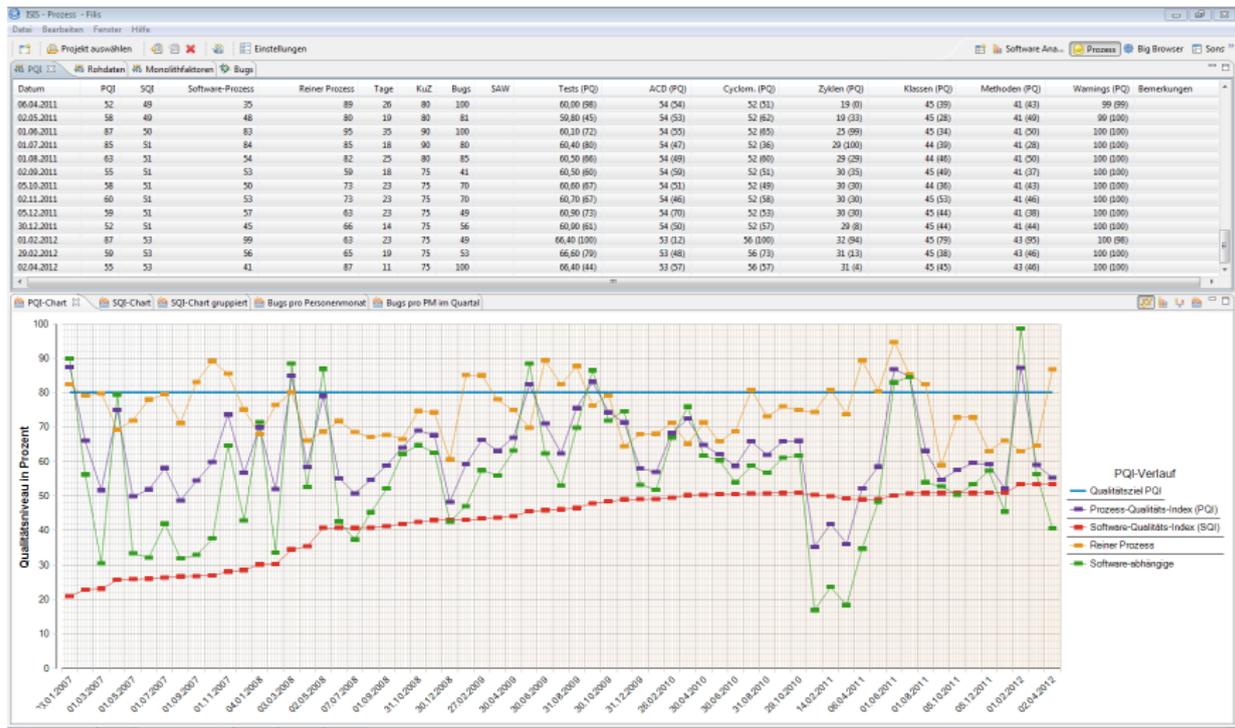
Weniger sinnvolle Metriken

Lines of Code (LOC)

- ▶ Problem der eindeutigen Definition
 - ▶ Kommentarzeilen
 - ▶ Zeilenumbrüche
- ▶ LOC und Funktionsumfang stehen in keinem direkten Verhältnis
 - ▶ Code-Duplizierung (Copy & Paste)
 - ▶ ineffiziente Implementierung
 - ▶ Refactoring kann Wert bei gleichem Funktionsumfang verringern
- ▶ Konsequenz: keine wirkliche Aussagekraft für Qualität

Aber: Anzahl Codezeilen pro Methode ist eine durchaus sinnvolle Metrik!

ISIS - Es werde Licht



USUS - Wo ist der Hund begraben?

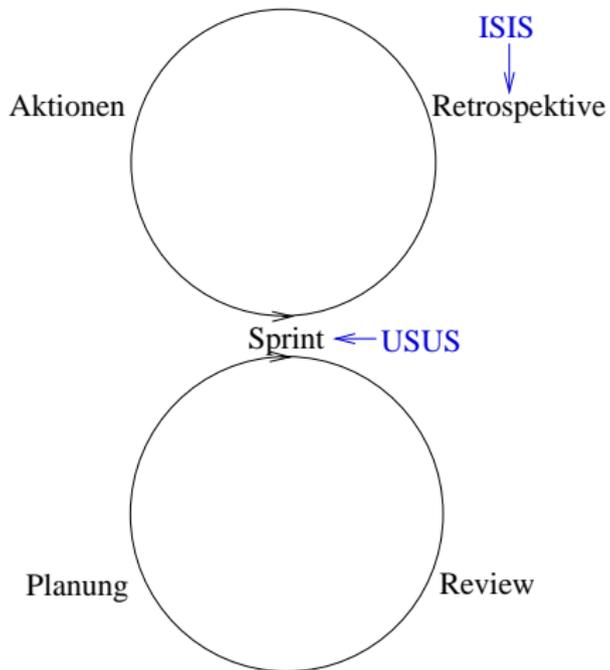
The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure for 'org.projectus.usus', including packages like 'org.projectus.usus.adapter', 'org.projectus.usus.core', and 'org.projectus.usus.core.internal.proportions.rawdata'.
- Code Editor:** Displays the source code for 'FileRelationClassDescriptorTest.java', showing a test method 'threeClassDescriptorsTwoRelationsFirstAndSecondRemovedInBetween()' and a 'checkTwo()' method.
- Class Graph:** A dependency graph showing relationships between packages like 'org.projectus.usus.core', 'org.projectus.usus.core.internal.proportions.rawdata', and 'org.projectus.usus.core.internal'.
- Usus Cockpit:** A dashboard showing various indicators and their values.

Indicator	Avg. Rating	Hotspots	Total	Trend
Code proportions				
Average class/method dependency	10.4	127	297 classes	
Class size	5.4	29	297 classes	
Cyclomatic complexity	0.3	12	1751 methods	
Method length	1.3	45	1751 methods	
Number of non-static, non-final public fields	2.7	8	297 classes	
Packages with cyclic dependencies	25.9	15	58 packages	

Value	Name	Path	Trend
17	FileRelationClassDescriptorTest	/org.projectus.usus.core/test/org/projectus.usus.core/relations/model/test	-1
41	ACCCollectorPDFTest	/org.projectus.usus.core/test/pdftest/org/projectus.usus.core/internal/proportions/rawdata/ccl	0
40	DependencyGraphView	/org.projectus.usus.ui.dependencygraph/src/org/projectus.usus.ui/dependencygraph/common	0
31	ClassDescriptor	/org.projectus.usus/src/org/projectus.usus.core/relations/model	0
23	NodeLabelProvider	/org.projectus.usus.ui.dependencygraph/src/org/projectus.usus.ui/dependencygraph/common	0
23	PackageNameNodeFilter	/org.projectus.usus.ui.dependencygraph/src/org/projectus.usus.ui/dependencygraph/filters	0
19	BoundType	/org.projectus.usus.core/src/org/projectus.usus.core/relations/model	0
19	CockpitView	/org.projectus.usus.ui/src/org/projectus.usus.ui/internal/proportions/cockpit	0
19	PackageNodeProvider	/org.projectus.usus.ui.dependencygraph/src/org/projectus.usus.ui/dependencygraph/filters	0

SCRUM - So bringt man es unter



Integration in Prozess

Continuous Build Server vs Integration in IDE

Continuous Build (ISIS)

- ▶ Historisierung der Änderungen
- ▶ langfristiger Erfolg wird sichtbar

Integration in IDE (USUS)

- ▶ Entwickler kann Problemstellen lokalisieren
- ▶ Entwickler sieht Erfolg der Änderung

Integration in Prozess

Broken Window Effect

Visualisierung der Metriken

- ▶ Erfolg messbar
- ▶ Erfolg sofort sichtbar
- ▶ ISIS: Historisierung
- ▶ USUS: sofortiger Erfolg
- ▶ Burndown-Chart: Fortschritt

Integration in Prozess

Maßnahmen

- ▶ TDD + PP + CI + XP einsetzen
- ▶ Refactoring gezielt einsetzen
- ▶ Code-Konventionen nutzen
- ▶ Retrospektive: Probleme adressieren

Metriken

Was bringen sie?

- ▶ Verstand benutzen
- ▶ Spaß beim Entwickeln
- ▶ Feedback für den Entwickler!

USUS

Weitere Informationen und Download

- ▶ USUS als Eclipse-Plugin
- ▶ Open Source: Eclipse Public License 1.0
- ▶ URL: <http://code.google.com/p/projectusus/>